INFORMATION AND CONTROL 16, 443–464 (1970)

The Parsing for General Phrase-Structure Grammars

JACQUES LOECKX

*MBLE Research Laboratory, Brussels, Belgium**

Received May 29, 1969; revised August 28, 1969

Pushdown automata serve as a base for the description of four basic parsing techniques for general phrase-structure grammars (Chomsky O-type grammars).

1. INTRODUCTION

Parsers for general phrase-structure grammars have been proposed by Griffiths [1, 2] and by Eickel and Paul [3].

On the other hand, parsers for context-free grammars have been described abundantly in the literature. A classification of these parsers into four classes is given by several authors (e.g., Cheatham and Sattley [4], Knuth [5], Feldman and Gries [6]). This classification distinguishes between top-down and bottom-up parsers and between trial-and-error and selective parsers but its description is informal and lacks precision; for instance, the parser described in Ref. [7] is considered as being of the top-down type in Ref. [6] and as being of the bottom-up type in Ref. [4].

In this paper four parsing techniques for general phrase-structure grammars are described with the help of two-pushdown automata, and the implementation of these techniques is discussed. The parsers for general phrase-structure grammars described in the literature appear to be particular implementations of two of these techniques; the other two parsing techniques give rise to essentially different parsers. Furthermore the distinction between the four parsing techniques corresponds to a generalization of the classification of parsers for context-free grammars; as a difference the use of two-pushdown automata provides a formal and, consequently, a precise definition. Finally, it is shown that the construction of selective parsers may be based on the

* Author's present address: Prof. Dr. J. Loeckx, Applied Mathematics Department, Twente University of Technology, PB 217, Enschede, Netherlands.

use of two-pushdown automata similar to the two-pushdown automata introduced here.

The paper presents moreover a definition of the syntactical structure of a sentence which is believed to be better suited for the description of a parser than those described elsewhere (Griffiths [2], Langmaack and Eickel [8]). It also contains a criticism of the papers of Griffiths and Petrick in which the efficiency of different parsers is compared [9, 10].

In Sections 2 and 3, the formal definitions of general phrase-structure grammars and two-pushdown automata are recalled. In Section 4, two acceptors are described which define the same language as a given grammar. In Section 5, four parsing techniques are deduced from these acceptors and the implementation of these techniques is discussed. Section 6 compares the efficiency of the parsing techniques described and indicates how two-pushdown automata may be used in the construction of selective parsers. Throughout the paper the simplifications occurring in the particular case of context-free grammars are mentioned.

Emphasis is laid on clarity and precision in the description but formal proofs are omitted.

2. Definitions Concerning Phrase-Structure Grammars

2.1. *General Phrase-Structure Grammar* (Chomsky [11], Aho and Ullman [12])

A *general phrase-structure grammar* (or *Chomsky O-type grammar*) is defined by a 4-tuple (V, T, R, Z). In this 4-tuple

V is a finite set of symbols called *vocabulary*,

T is a subset of V called *terminal vocabulary*,

R is a finite set of ordered pairs $\phi \rightarrow \psi$ with ϕ in $V^* - \{\epsilon\}$ and ψ in V^*;[1] an element $\phi \rightarrow \psi$ of R is called a *rewriting rule*, ϕ and ψ are called respectively *left* and *right member* of the rewriting rule,

Z is an element of $V - T$ called *initial symbol.*

See Fig. 1.

A *sentence* is a string x in T^* for which there exist a finite sequence

$$\omega_1 , \omega_2 ,..., \omega_{n-1} , \omega_n \qquad (n > 1)$$

[1] V^* denotes the set of all finite sequences, called strings, of elements from V, including the empty string; ϵ denotes the empty string.

$V = \{Z, D, a, b, c\}$
$T = \{a, b, c\}$
$Z = Z$
R consists of the following 4 rewriting rules:

$$Z \to aZD$$
$$Z \to abc$$
$$bD \to bbc$$
$$cD \to Dc$$

FIG. 1. A general phrase-structure grammar (V, T, R, Z). It defines the language $\{a^n b^n a^n \mid n > 0\}$.

and strings σ_i, τ_i, ϕ_i and ψ_i in V^* $(1 \leqslant i \leqslant n-1)$ such that

$$\omega_1 = Z, \tag{1}$$

$$\omega_n = x, \tag{2}$$

$$\omega_i = \sigma_i \phi_i \tau_i \qquad (1 \leqslant i \leqslant n-1), \tag{3}$$

$$\omega_{i+1} = \sigma_i \psi_i \tau_i \qquad (1 \leqslant i \leqslant n-1), \tag{4}$$

and

$$\phi_i \to \psi_i \quad \text{is in } R \qquad (1 \leqslant i \leqslant n-1). \tag{5}$$

Intuitively, a sentence is a string of terminal symbols obtained from the initial symbol in a certain number of generation steps, each generation step consisting in the application of a rewriting rule. The sequence

$$\omega_1, \omega_2, \ldots, \omega_{n-1}, \omega_n$$

is called a *generation sequence* of the sentence; the same sequence written inversely, viz., as

$$\omega_n, \omega_{n-1}, \ldots, \omega_2, \omega_1$$

is called a *reduction sequence*. See Figs. 2 and 3.

The *language* defined by a phrase-structure grammar is the set of its sentences.

2.2. *Syntactical Structure of a Sentence*

Let (V, T, R, Z) be a general phrase-structure grammar and x one of its sentences. The notions of generation structure, reduction structure and syntactical graph of this sentence will now be defined.

Z	Z
aZD	(aZD)
$aaZDD$	$a(aZD)D$
$aaaZDDD$	$aa(aZD)DD$
$aaaabcDDD$	$aaa(abc)DDD$
$aaaabDcDD$	$aaaab(Dc)DD$
$aaaabbccDD$	$aaaa(bbc)cDD$
$aaaabbcDcD$	$aaaabbc(Dc)D$
$aaaabbDccD$	$aaaabb(Dc)cD$
$aaaabbbcccD$	$aaaab(bbc)ccD$
$aaaabbbccDc$	$aaaabbbcc(Dc)$
$aaaabbbcDcc$	$aaaabbbc(Dc)c$
$aaaabbbDccc$	$aaaabbb(Dc)cc$
$aaaabbbbcccc$	$aaaabb(bbc)ccc$

FIG. 2. The left column constitutes a generation sequence of the unambiguous sentence *aaaabbbbcccc* of the language defined by the grammar of Fig. 1; this generation sequence is the canonical one. The right column constitutes the generation structure of the same sentence.

$aaaabbbbcccc$	$aaaabbbbcccc$
$aaaabbbDccc$	$aaaabb(bD)ccc$
$aaaabbbcDcc$	$aaaabbb(cD)cc$
$aaaabbDDcc$	$aaaab(bD)Dcc$
$aaaabbDcDc$	$aaaabbD(cD)c$
$aaaabbcDDc$	$aaaabb(cD)Dc$
$aaaabDDDc$	$aaaa(bD)DDc$
$aaaabDDcD$	$aaaabDD(cD)$
$aaaabDcDD$	$aaaabD(cD)D$
$aaaabcDDD$	$aaaab(cD)DD$
$aaaZDDD$	$aaa(Z)DDD$
$aaZDD$	$aa(Z)DD$
aZD	$a(Z)D$
Z	(Z)

FIG. 3. The left column constitutes a reduction sequence of the sentence *aaaabbbbcccc* of Fig. 2; this reduction sequence is the canonical one. The right column constitutes the reduction structure of the same sentence.

Suppose that

$$\omega_1 , \omega_2 ,..., \omega_{n-1} , \omega_n$$

is a generation sequence of the sentence x, i.e., a sequence such that there exist strings $\sigma_i , \tau_i , \phi_i$ and ψ_i in V^* satisfying the conditions (1) through (5) of Section 2.1. This generation sequence is called *canonical* if there exist moreover nonempty strings ρ_i in $V^* - \{\epsilon\}$ $(1 \leqslant i \leqslant n - 2)$ such that

$$\psi_i\tau_i = \rho_i\tau_{i+1} \qquad (1 \leqslant i \leqslant n - 2). \tag{6}$$

Intuitively, these supplementary conditions (6) express that the symbols rewritten during the $(i + 1)$-th generation step are not completely to the left of those rewritten during the i-th generation step. Suppose that

$$\omega_1 , \omega_2 ,..., \omega_{n-1} , \omega_n$$

is such a canonical generation sequence. By definition the sequence

$$\omega_1', \omega_2',..., \omega_{n-1}' , \omega_n'$$

with

$$\omega_1' = \omega_1$$

and

$$\omega_{i+1}' = \sigma_i(\psi_i)\tau_i \qquad (1 \leqslant i \leqslant n - 1),$$

is called a *generation structure* of the sentence x.[2] It is easily shown that a sentence possesses at least one generation structure; actually, the notion of generation structure is equivalent with that of canonical derivation introduced in Griffiths [2] or that of phrase structure introduced in Langmaack and Eickel [8] but is believed to be better adapted to the description of parsers. See Fig. 2.

Similarly, a reduction sequence

$$\omega_n , \omega_{n-1} ,..., \omega_2 , \omega_1$$

is canonical if in addition to the conditions (1) through (5) the following condition is satisfied: there exist ρ_i in $V^* - \{\epsilon\}$ $(1 \leqslant i \leqslant n - 2)$ such that

$$\sigma_i\psi_i = \sigma_{i+1}\rho_i \qquad (1 \leqslant i \leqslant n - 2). \tag{6'}$$

[2] The parentheses (and) are supposed not to be symbols of V.

Intuitively, these conditions (6′) express that the symbols rewritten during the $(n - i)$-th reduction step are not completely to the left of those rewritten during the $(n - i - 1)$-th reduction step. If

$$\omega_n , \omega_{n-1} ,..., \omega_2 , \omega_1$$

is such a canonical reduction sequence, the sequence

$$\omega_n', \omega_{n-1}', ..., \omega_2', \omega_1'$$

with

$$\omega_n' = \omega_n$$

and

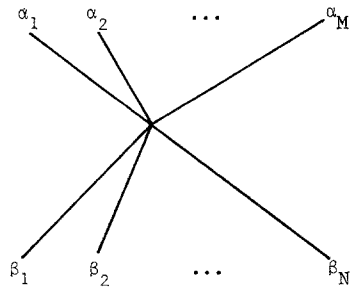
$$\omega_i' = \sigma_i(\phi_i)\, \tau_i \qquad (1 \leqslant i \leqslant n - 1),$$

is called a *reduction structure* of the sentence. See Fig. 3.

Finally, a *syntactical graph* of the sentence x may be defined informally as follows. To each rewriting rule

$$\alpha_1\alpha_2 \cdots \alpha_M \rightarrow \beta_1\beta_2 \cdots \beta_N \qquad (\alpha_1 ,..., \alpha_M , \beta_1 ,..., \beta_N \text{ in } V;\ M \geqslant 1;\ N \geqslant 0)$$

applied during the generation of x corresponds the following branching with M upper limbs and N lower limbs:



the M upper limbs point to M nodes valued $\alpha_1 , \alpha_2 ,..., \alpha_M$ and the N lower limbs to N nodes valued $\beta_1 , \beta_2 ,..., \beta_N$. See Fig. 4.

It is easily shown that the set of generation structures, the set of reduction structures and the set of syntactical graphs are in a one-to-one correspondence and that there exist simple algorithms which map an element of one of these sets into the corresponding element of another of these sets. Hence the generation structure, the reduction structure and the syntactical graph

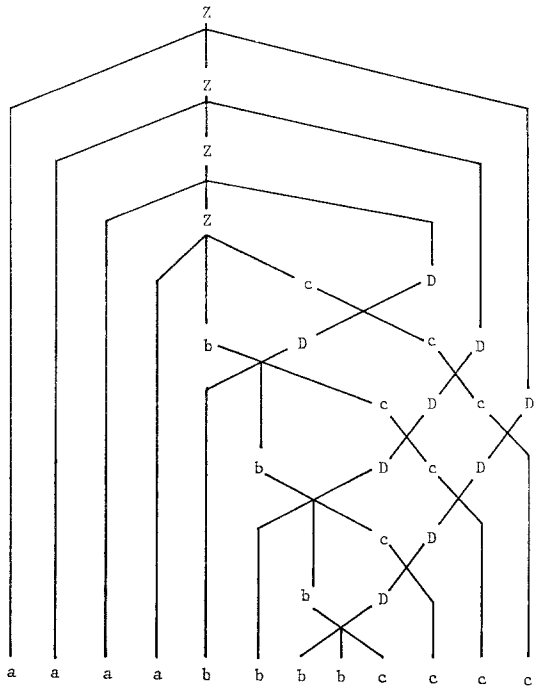


FIG. 4. The syntactical graph of the sentence *aaaabbbbcccc* of Fig. 2.

may be considered as three different descriptions of a single concept called the *syntactical structure* of a sentence.

It is easily seen that the canonical generation sequence and the canonical reduction sequence also constitute descriptions of the syntactical structure when the grammar satisfies the following (sufficient but not necessary) condition: for any two rewriting rules $\phi_1 \rightarrow \psi_1$ and $\phi_2 \rightarrow \psi_2$ there does not exist λ, μ, λ' and μ' in V^* such that

$$\lambda\phi_1\mu = \lambda'\phi_2\mu'$$

and

$$\lambda\psi_1\mu = \lambda'\psi_2\mu'.^3$$

[3] The condition is illustrated by the following example: If ABC and ADC are two successive strings in a canonical generation sequence and $AB \rightarrow AD$ and $BC \rightarrow DC$ are two rewriting rules, the canonical generation sequence does not indicate which of these rewriting rules has been applied to the string ABC (whereas the generation structure does).

Though this condition is satisfied for most grammars (and in particular for the grammar of Fig. 1) the general case will be considered throughout the paper; more precisely, the syntactical structure will be described either by the generation structure or the reduction structure.

2.3. *Ambiguity and Parsing*

A sentence is *ambiguous* if it possesses more than one syntactical structure. A language is *ambiguous* if it contains at least one ambiguous sentence.

A *parser* for a general phrase-structure grammar is a function—or, equivalently, an algorithm—mapping the sentences of the language defined by the grammar into their syntactical structures; the value of this function is undefined for strings which are not sentences.[4]

2.4. *The particular Case of Context-Free Grammars* (Ginsburg [13]).

A *context-free grammar* is a general phrase-structure grammar each rewriting rule of which is of the form,

$$A \rightarrow \phi,$$

with A in $V - T$ and ϕ in V^*.

The definitions of the previous Sections greatly reduce. In particular, the $n - 2$ conditions (6) of Section 2.2 may be replaced by the $n - 2$ conditions,

$$\sigma_{i+1} \text{ is in } T^* \qquad (1 \leqslant i \leqslant n - 2).$$

Moreover, the condition of Section 2.2 is always satisfied; hence, the canonical generation sequence and the canonical reduction sequence may always be used instead of the generation structure and the reduction structure respectively.

3. Definitions Concerning Two-Pushdown Automata

3.1. *Two-Pushdown Acceptors*

Intuitively, a two-pushdown acceptor is obtained by adding a supplementary pushdown tape to a classical one-pushdown acceptor (e.g., Chomsky [14], Ginsburg [13]).

[4] In practical applications it is interesting to map nonsentences into an "error indication"; this is not possible in the general case as the language defined by a general phrase-structure grammar is not necessarily recursive (Chomsky [11]).

A physical model for a two-pushdown acceptor is on Fig. 5. Two-pushdown acceptors are known to be equivalent with nondeterministic Turing acceptors (Evey [15], Fischer [16]); they are similar to one-way list-storage acceptors (Ginsburg and Harrison [17]).

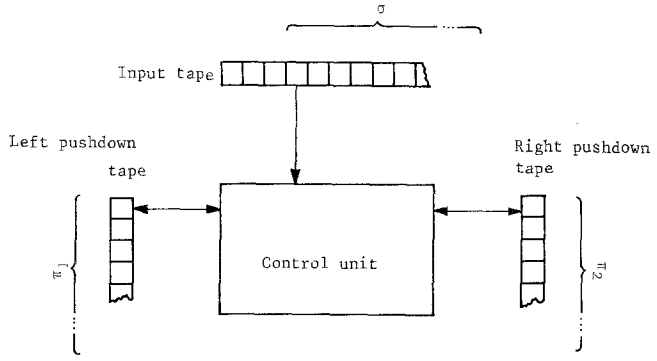


FIG. 5. A physical model for a two-pushdown acceptor. If the control unit is in the state q, the configuration of the acceptor is $(q, \pi_1, \pi_2, \sigma)$.

Formally, a (*nondeterministic*) *two-pushdown acceptor* is defined by a 6-tuple (S, P, Q, I, q_s, q_f). In this 6-tuple[5]

(1) S, P and Q are finite sets of symbols called *input symbols*, *pushdown symbols* and *states* respectively,

(2) I is a finite set of 7-tuples called *instructions* which are written

$$(q, \pi_1, \pi_2, \sigma) \rightarrow (q', \pi_1', \pi_2')$$

with q and q' in Q, π_1, π_2, π_1' and π_2' in P^* and σ in S^*,

(3) q_s and q_f are elements of Q and are called the *start state* and *final state*, respectively.

See Figs. 6 and 8.

Let $A = (S, P, Q, I, q_s, q_f)$ be a two-pushdown acceptor. A *configuration* of this acceptor is a 4-tuple $(q, \pi_1, \pi_2, \sigma)$ with q in Q, π_1 and π_2 in P^* and σ in S^*. In the physical model q corresponds to the state of the control unit, π_1 and π_2 to the contents of the left and right pushdown tapes and σ to the contents of the fraction of the input tape which has not been read yet.

[5] Cf. the *generalized pda* introduced in Ginsburg [13].

$S = \{a, b, c\}$
$P = \{Z, D, a, b, c, \#\}$
$Q = \{q_s, q_f, q\}$
I consists of the following instructions:

$$(q_s, \epsilon, \epsilon, \epsilon) \rightarrow (q, \#, Z\#) \quad (1)$$
$$(q, \epsilon, Z, \epsilon) \rightarrow (q, \epsilon, aZD) \quad (2a)$$
$$(q, \epsilon, Z, \epsilon) \rightarrow (q, \epsilon, abc) \quad (2b)$$
$$(q, b, D, \epsilon) \rightarrow (q, \epsilon, bbc) \quad (2c)$$
$$(q, c, D, \epsilon) \rightarrow (q, \epsilon, Dc) \quad (2d)$$
$$(q, \epsilon, b, \epsilon) \rightarrow (q, b, \epsilon) \quad (3a)$$
$$(q, \epsilon, c, \epsilon) \rightarrow (q, c, \epsilon) \quad (3b)$$
$$(q, \#, a, a) \rightarrow (q, \#, \epsilon) \quad (4a)$$
$$(q, \#, b, b) \rightarrow (q, \#, \epsilon) \quad (4b)$$
$$(q, \#, c, c) \rightarrow (q, \#, \epsilon) \quad (4c)$$
$$(q, \#, \#, \epsilon) \rightarrow (q_f, \epsilon, \epsilon) \quad (5)$$

FIG. 6. The two-pushdown acceptor (S, P, Q, I, q_s, q_f). This acceptor is the top-down acceptor associated with the grammar of Fig. 1; only those instructions $(q, \epsilon, \alpha, \epsilon) \rightarrow (q, \alpha, \epsilon)$ of type (3) are listed for which there exist ϕ and ω in V^* and ψ in $V^* - \{\epsilon\}$ such that $\phi\alpha\psi \rightarrow \omega$ is in R; in fact, the other instructions of type (3) are superfluous.

See Fig. 5; note that the top of the left pushdown tape corresponds to the rightmost symbol of π_1, whereas the top of the right pushdown tape corresponds to the leftmost symbol of π_2.

By definition, the relation,

$$(q, \pi_1, \pi_2, \sigma) \models (q', \pi_1', \pi_2', \sigma'),$$

between two configurations holds with respect to the acceptor A if there exist π_{10}, π_{20}, π_{1a}, π_{2a}, π_{1b} and π_{2b} in P^* and σ_{1a} in S^* such that

$$\pi_1 = \pi_{10}\pi_{1a}, \qquad \pi_1' = \pi_{10}\pi_{1b},$$
$$\pi_2 = \pi_{2a}\pi_{20}, \qquad \pi_2' = \pi_{2b}\pi_{20},$$
$$\sigma = \sigma_{1a}\sigma',$$

and

$$(q, \pi_{1a}, \pi_{2a}, \sigma_{1a}) \rightarrow (q', \pi_{1b}, \pi_{2b}) \text{ is in } I.$$

On the physical model this relation represents the application of an instruction: π_{1a}, π_{2a} and σ_{1a} are read from the left pushdown tape, the right pushdown tape and the input tape respectively, the state q' is entered and

$(q_s, \epsilon, \epsilon, aaaabbbbcccc)$	apply (1)		
$(q, \#, Z\#, aaaabbbbcccc)$	apply (2a)	Z	Z
$(q, \#, aZD\#, aaaabbbbcccc)$	apply (4a)	aZD	(aZD)
$(q, \#, ZD\#, aaabbbbcccc)$	apply (2a)		
$(q, \#, aZDD\#, aaabbbbcccc)$	apply (4a)	$aaZDD$	$a(aZD)D$
$(q, \#, ZDD\#, aabbbbcccc)$	apply (2a)		
$(q, \#, aZDDD\#, aabbbbcccc)$	apply (4a)	$aaaZDDD$	$aa(aZD)DD$
$(q, \#, ZDDD\#, abbbbcccc)$	apply (2b)		
$(q, \#, abcDDD\#, abbbbcccc)$	apply (4a)	$aaaabcDDD$	$aaa(abc)DDD$
$(q, \#, bcDDD\#, bbbbcccc)$	apply (3a)		
$(q, \#b, cDDD\#, bbbbcccc)$	apply (3b)		
$(q, \#bc, DDD\#, bbbbcccc)$	apply (2d)		
$(q, \#b, DcDD\#, bbbbcccc)$	apply (2c)	$aaaabDcDD$	$aaaab(Dc)DD$
$(q, \#, bbccDD\#, bbbbcccc)$	apply (4b)	$aaaabbccDD$	$aaaa(bbc)cDD$
$(q, \#, bccDD\#, bbbcccc)$	apply (3a)		
$(q, \#b, ccDD\#, bbbcccc)$	apply (3b)		
$(q, \#bc, cDD\#, bbbcccc)$	apply (3b)		
$(q, \#bcc, DD\#, bbbcccc)$	apply (2d)		
$(q, \#bc, DcD\#, bbbcccc)$	apply (2d)	$aaaabbcDcD$	$aaaabbc(Dc)D$
$(q, \#b, DccD\#, bbbcccc)$	apply (2c)	$aaaabbDccD$	$aaaabb(Dc)cD$
$(q, \#, bbcccD\#, bbbcccc)$	apply (4b)	$aaaabbbcccD$	$aaaab(bbc)ccD$
$(q, \#, bcccD\#, bbcccc)$	apply (3a)		
$(q, \#b, cccD\#, bbcccc)$	apply (3b)		
$(q, \#bc, ccD\#, bbcccc)$	apply (3b)		
$(q, \#bcc, cD\#, bbcccc)$	apply (3b)		
$(q, \#bccc, D\#, bbcccc)$	apply (2d)		
$(q, \#bcc, Dc\#, bbcccc)$	apply (2d)	$aaaabbbccDc$	$aaaabbbcc(Dc)$
$(q, \#bc, Dcc\#, bbcccc)$	apply (2d)	$aaaabbbcDcc$	$aaaabbbc(Dc)c$
$(q, \#b, Dccc\#, bbcccc)$	apply (2c)	$aaaabbbDccc$	$aaaabbb(Dc)cc$
$(q, \#, bbcccc\#, bbcccc)$	apply (4b)	$aaaabbbbcccc$	$aaaabb(bbc)ccc$
$(q, \#, bcccc\#, bcccc)$	apply (4b)		
$(q, \#, cccc\#, cccc)$	apply (4c)		
$(q, \#, ccc\#, ccc)$	apply (4c)		
$(q, \#, cc\#, cc)$	apply (4c)		
$(q, \#, c\#, c)$	apply (4c)		
$(q, \#, \#, \epsilon)$	apply (5)		
$(q_f, \epsilon, \epsilon, \epsilon)$			

FIG. 7. The first column contains the sequence of configurations proving that $(q_s, \epsilon, \epsilon, aaaabbbbcccc) \overset{*}{\models} (q_f, \epsilon, \epsilon, \epsilon)$ holds with respect to the two-pushdown acceptor of Fig. 6, i.e., proving that the string *aaaabbbbcccc* belongs to the language defined by this acceptor; the instructions of Fig. 6 which are applied to these configurations are in the second column. The third column contains the strings associated with the configurations of the left column according to Section 4.1; these strings constitute the canonical generation sequence of the sentence *aaaabbbbcccc*. The fourth column contains the strings printed out according to Section 5.1; these strings constitute the generation structure of the sentence (cf. Fig. 2).

$S = \{a, b, c\}$
$P = \{Z, D, a, b, c, \#\}$
$Q = \{q_s, q_f, q\}$
I consists of the following instructions:

$$(q_s, \epsilon, \epsilon, \epsilon) \rightarrow (q, \#, \#) \quad (1')$$
$$(q, aZD, \epsilon, \epsilon) \rightarrow (q, Z, \epsilon) \quad (2'a)$$
$$(q, abc, \epsilon, \epsilon) \rightarrow (q, Z, \epsilon) \quad (2'b)$$
$$(q, bbc, \epsilon, \epsilon) \rightarrow (q, b, D) \quad (2'c)$$
$$(q, Dc, \epsilon, \epsilon) \rightarrow (q, c, D) \quad (2'd)$$
$$(q, \epsilon, D, \epsilon) \rightarrow (q, D, \epsilon) \quad (3')$$
$$(q, \epsilon, \#, a) \rightarrow (q, a, \#) \quad (4'a)$$
$$(q, \epsilon, \#, b) \rightarrow (q, b, \#) \quad (4'b)$$
$$(q, \epsilon, \#, c) \rightarrow (q, c, \#) \quad (4'c)$$
$$(q, \#Z, \#, \epsilon) \rightarrow (q_f, \epsilon, \epsilon) \quad (5')$$

FIG. 8. The two-pushdown acceptor (S, P, Q, I, q_s, q_f). This acceptor is the bottom-up acceptor associated with the grammar of Figure 1; only those instructions $(q, \epsilon, \alpha, \epsilon) \rightarrow (q, \alpha, \epsilon)$ of type (3') are listed for which there exist ψ and ω in V^* and ϕ in $V^* - \{\epsilon\}$ such that $\phi\alpha\psi \rightarrow \omega$ is in R.

finally π_{1b} and π_{2b} are written on the pushdown tapes. See Fig. 7 and Fig. 9: the relation $\models$ holds between the configurations of any two consecutive lines of the left column.

The relation $\overset{*}{\models}$ is the reflexive-transitive closure of the relation $\models$. On the physical model this relation corresponds to a "run" of the acceptor.

The *language defined by a two-pushdown acceptor* is the set of all strings x for which

$$(q_s, \epsilon, \epsilon, x) \overset{*}{\models} (q_f, \epsilon, \epsilon, \epsilon)$$

holds with respect to the acceptor, i.e., the set

$$\{x \mid (q_s, \epsilon, \epsilon, x) \overset{*}{\models} (q_f, \epsilon, \epsilon, \epsilon)\}.$$

See Figs. 7 and 9.

A two-pushdown acceptor is called *deterministic* if at most one instruction is applicable to any given configuration.

3.2. *Two-Pushdown Transducers*

Intuitively a two-pushdown transducer is obtained by adding an output tape to a two-pushdown acceptor. Its formal definition is easily deduced from those given above.

$(q_s, \epsilon, \epsilon, aaaabbbbcccc)$	apply (1′)		
$(q, \#, \#, aaaabbbbcccc)$	apply (4′a)	*aaaabbbbcccc*	*aaaabbbbcccc*
$(q, \#a, \#, aaabbbbcccc)$	apply (4′a)		
$(q, \#aa, \#, aabbbbcccc)$	apply (4′a)		
$(q, \#aaa, \#, abbbbcccc)$	apply (4′a)		
$(q, \#aaaa, \#, bbbbcccc)$	apply (4′b)		
$(q, \#aaaab, \#, bbbcccc)$	apply (4′b)		
$(q, \#aaaabb, \#, bbcccc)$	apply (4′b)		
$(q, \#aaaabbb, \#, bcccc)$	apply (4′b)		
$(q, \#aaaabbbb, \#, cccc)$	apply (4′c)		
$(q, \#aaaabbbbc, \#, ccc)$	apply (2′c)		
$(q, \#aaaabbb, D\#, ccc)$	apply (3′)	*aaaabbbDccc*	*aaaabb*(*bD*)*ccc*
$(q, \#aaaabbbD, \#, ccc)$	apply (4′c)		
$(q, \#aaaabbbDc, \#, cc)$	apply (2′d)		
$(q, \#aaaabbbc, D\#, cc)$	apply (2′c)	*aaaabbbcDcc*	*aaaabbb*(*cD*)*cc*
$(q, \#aaaabb, DD\#, cc)$	apply (3′)	*aaaabbDDcc*	*aaaab*(*bD*)*Dcc*
$(q, \#aaaabbD, D\#, cc)$	apply (3′)		
$(q, \#aaaabbDD, \#, cc)$	apply (4′c)		
$(q, \#aaaabbDDc, \#, c)$	apply (2′d)		
$(q, \#aaaabbDc, D\#, c)$	apply (2′d)	*aaaabbDcDc*	*aaaabbD*(*cD*)*c*
$(q, \#aaaabbc, DD\#, c)$	apply (2′c)	*aaaabbcDDc*	*aaaabb*(*cD*)*Dc*
$(q, \#aaaab, DDD\#, c)$	apply (3′)	*aaaabDDDc*	*aaaa*(*bD*)*DDc*
$(q, \#aaaabD, DD\#, c)$	apply (3′)		
$(q, \#aaaabDD, D\#, c)$	apply (3′)		
$(q, \#aaaabDDD, \#, c)$	apply (4′c)		
$(q, \#aaaabDDDc, \#, \epsilon)$	apply (2′d)		
$(q, \#aaaabDDc, D\#, \epsilon)$	apply (2′d)	*aaaabDDcD*	*aaaabDD*(*cD*)
$(q, \#aaaabDc, DD\#, \epsilon)$	apply (2′d)	*aaaabDcDD*	*aaaabD*(*cD*)*D*
$(q, \#aaaabc, DDD\#, \epsilon)$	apply (2′b)	*aaaabcDDD*	*aaaab*(*cD*)*DD*
$(q, \#aaaZ, DDD\#, \epsilon)$	apply (3′)	*aaaZDDD*	*aaa*(*Z*)*DDD*
$(q, \#aaaZD, DD\#, \epsilon)$	apply (2′a)		
$(q, \#aaZ, DD\#, \epsilon)$	apply (3′)	*aaZDD*	*aa*(*Z*)*DD*
$(q, \#aaZD, D\#, \epsilon)$	apply (2′a)		
$(q, \#aZ, D\#, \epsilon)$	apply (3′)	*aZD*	*a*(*Z*)*D*
$(q, \#aZD, \#, \epsilon)$	apply (2′a)		
$(q, \#Z, \#, \epsilon)$	apply (5′)	*Z*	(*Z*)
$(q_f, \epsilon, \epsilon, \epsilon)$			

FIG. 9. The first column contains the sequence of configurations proving that $(q_s, \epsilon, \epsilon, aaaabbbbcccc) \overset{*}{\vdash} (q_f, \epsilon, \epsilon, \epsilon)$ holds with respect to the pushdown acceptor of FIG. 8, i.e., proving that the string *aaaabbbbcccc* belongs to the language defined by this acceptor; the instructions of FIG. 8 which are applied to these configurations are in the second column. The third column contains the strings associated with the configurations of the left column according to Section 4.2; these strings constitute the canonical reduction sequence of the sentence *aaaabbbbcccc*. The fourth column contains the strings printed out according to Section 5.1; these strings constitute the reduction structure of the sentence (cf. Fig. 3).

A nondeterministic, respectively deterministic, two-pushdown transducer defines a multivalued, respectively univalued, partially computable function (e.g., Evey [15]).

4. Two acceptors for Phrase-Structure Languages

Let $G = (V, T, R, Z)$ be an arbitrary general phrase-structure grammar and $\#$ a symbol not contained in V. Two nondeterministic two-pushdown acceptors will now be formally described which define the same language as G. As already indicated, these acceptors will constitute a basis for the description of parsing techniques in Section 5.

4.1. *The Top-Down Acceptor*

The *top-down acceptor* associated with the grammar G is the two-pushdown acceptor (S, P, Q, I, q_s, q_f) with $S = T$, $P = V \cup \{\#\}$, $Q = \{q_s, q_f, q\}$, and I consisting of the following instructions:

$$(q_s, \epsilon, \epsilon, \epsilon) \rightarrow (q, \#, Z\#), \tag{1}$$

$$(q, \phi, \alpha, \epsilon) \rightarrow (q, \epsilon, \psi) \tag{2}$$

for each ϕ and ψ in V^* and α in V such that $\phi\alpha \rightarrow \psi$ is in R,

$$(q, \epsilon, \alpha, \epsilon) \rightarrow (q, \alpha, \epsilon) \qquad \text{for each } \alpha \text{ in } V, \tag{3}$$

$$(q, \#, a, a) \rightarrow (q, \#, \epsilon) \qquad \text{for each } a \text{ in } T, \tag{4}$$

and

$$(q, \#, \#, \epsilon) \rightarrow (q_f, \epsilon, \epsilon). \tag{5}$$

It can be easily proved that the language defined by this acceptor, viz.,

$$\{x \mid (q_s, \epsilon, \epsilon, x) \overset{*}{\vdash} (q_f, \epsilon, \epsilon, \epsilon)\}$$

is exactly the language defined by the grammar G. In fact, it is intuitively clear that the instruction (1) is applied at the start; each instruction of type (2) corresponds to a generation step, i.e., to the application of a rewriting rule; the instructions of type (3) prepare for the application of a rewriting rule with a left member consisting of more than one symbol; the instructions of type (4) check the equality of the symbols generated and those on the input tape; finally, the instruction (5) stops the acceptor. See Figs. 6 and 7.

The working of the acceptor is also illustrated by its relation with the canonical generation sequences of the sentences. Let

$$(q, \#\pi_1 , \pi_2\#, \sigma)$$

be a configuration for which there exist σ_0 in S^*, ϕ, ψ and π_{20} in P^* and α in P such that

$$(q_s , \epsilon, \epsilon, \sigma_0\sigma) \models^* (q, \#\pi_1\phi, \alpha\pi_{20}\#, \sigma) \models (q, \#\pi_1 , \pi_2\#, \sigma) \models^* (q_f , \epsilon, \epsilon, \epsilon)$$

and

$$\pi_2 = \psi\pi_{20} .$$

Less formally, $(q, \#\pi_1 , \pi_2\#, \sigma)$ is a configuration resulting from the application of the instruction

$$(q, \phi, \alpha, \epsilon) \rightarrow (q, \epsilon, \psi)$$

of type (2) and σ_0 is the fraction of the sentence $\sigma_0\sigma$ on the input tape which has already been read in. It is then easily seen that the following property holds: the sequence of strings obtained by associating with the configurations resulting from the application of an instruction of type (1) or type (2) the strings Z and $\sigma_0\pi_1\pi_2$ respectively constitutes the canonical generation sequence of the sentence. See Fig. 7.

It may be noted that the working of the top-down acceptor is similar to that of the two-pushdown-tape nondeterministic Turing machine introduced in Griffiths [2].

4.2. *The Bottom-Up Acceptor*

The *bottom-up acceptor* associated with the grammar G is the two-pushdown acceptor (S, P, Q, I, q_s , q_f) with S, P, Q defined as for the top-down acceptor and I consisting of the following instructions:

$$(q_s , \epsilon, \epsilon, \epsilon) \rightarrow (q, \#, \#), \tag{1'}$$

$$(q, \psi, \epsilon, \epsilon) \rightarrow (q, \alpha, \phi) \tag{2'}$$

for each ϕ and ψ in V^* and α in V such that $\alpha\phi \rightarrow \psi$ is in R,

$$(q, \epsilon, \alpha, \epsilon) \rightarrow (q, \alpha, \epsilon) \qquad \text{for each } \alpha \text{ in } V, \tag{3'}$$

$$(q, \epsilon, \#, a) \rightarrow (q, a, \#) \qquad \text{for each } a \text{ in } T, \tag{4'}$$

$$(q, \#Z, \#, \epsilon) \rightarrow (q_f , \epsilon, \epsilon). \tag{5'}$$

Again, it can be easily proved that the language defined by this acceptor, viz.,

$$\{x \mid (q_s , \epsilon, \epsilon, x) \overset{*}{\models} (q_f , \epsilon, \epsilon, \epsilon)\}$$

is exactly the language defined by the grammar G. In fact, it is intuitively clear that the instruction (1′) is applied at the start; each instruction of type (2′) corresponds to a reduction step, i.e., to the inverse application of a rewriting rule; the instructions of type (3′) care for the symbols introduced by the inverse application of a rewriting rule with a left member consisting of more than one symbol; the instructions of type (4′) read in a symbol from the input tape; finally, the instruction (5′) checks that the initial symbol Z has been obtained and stops the acceptor. See Figs. 8 and 9.

The working of the acceptor is also illustrated by its relation with the canonical reduction sequence. Let

$$(q, \#\pi_1 , \pi_2\#, \sigma)$$

be a configuration for which there exist σ_0 in S^*, ϕ, ψ, π_{10} and π_{20} in P^* and α in P such that

$$(q_s , \epsilon, \epsilon, \sigma_0\sigma) \overset{*}{\models} (q, \#\pi_{10}\psi, \pi_{20}\#, \sigma) \models (q, \#\pi_1 , \pi_2\#, \sigma) \overset{*}{\models} (q_f , \epsilon, \epsilon, \epsilon),$$

$$\pi_1 = \pi_{10}\alpha,$$

and

$$\pi_2 = \phi\pi_{20} .$$

Less formally, $(q, \#\pi_1 , \pi_2\#, \sigma)$ is a configuration resulting from the application of the instruction

$$(q, \psi, \epsilon, \epsilon) \rightarrow (q, \alpha, \phi)$$

of type (2′) and σ_0 is the fraction of the sentence $\sigma_0\sigma$ on the input tape already read in. The sequence of strings obtained by associating with the configurations resulting from the application of an instruction of type (1′) or (2′) the sentence and the strings $\pi_1\pi_2\sigma$ respectively constitutes the canonical reduction sequence of the sentence. See Fig. 9.

4.3. *The Particular Case of Context-Free Grammars*

The left pushdown tape of the top-down acceptor is useless when the grammar is context-free; in fact, its contents is constantly the single symbol #. Hence, the left pushdown tape may be deleted, i.e., the two-pushdown acceptor may be replaced by a one-pushdown acceptor. Actually, the

top-down one-pushdown acceptor so obtained is well-known (e.g., Ginsburg [13]).

Similarly, the right pushdown tape of the bottom-up acceptor is useless when the grammar is context-free. Hence, this two-pushdown acceptor may also be replaced by a one-pushdown acceptor.

5. Four Parsing Techniques for General Phrase-Structure Grammars

In Section 4, two two-pushdown acceptors have been described which define the same language as a given general phrase-structure grammar: the top-down acceptor and the bottom-up acceptor. Clearly, each of them is turned into a parser for this grammar if the two following modifications are performed. First, the acceptor has to be replaced by a transducer outputting the syntactical structures. Second, the nondeterministic transducer thus obtained has to be replaced by a deterministic one or, equivalently, by an algorithm. Both modifications will now be described.

5.1. *The First Modification*

Consider the two-pushdown transducer identical with the top-down acceptor except for the following: each time an instruction of type (1) or type (2) is applied a string is written on the output tape. In the case of the instruction of type (1) this string consists of the symbol Z; in the case of an instruction of type (2) it consists of the string,

$$\sigma_0\pi_1(\psi)\,\pi_{20}\,,$$

where σ_0, π_1, ψ, and π_{20} are defined as in Section 4.1. It is clear that this transducer exactly maps the sentences into their generation structures. See Fig. 7.

Similarly, consider the two-pushdown transducer identical with the bottom-up acceptor except for printing a string each time an instruction of type (1′) or type (2′) is applied. In the case of the instruction of type (1′) this string is the string contained by the input tape, i.e., the sentence to be parsed; in the case of an instruction of type (2′) it is the string,

$$\pi_{10}(\alpha\phi)\,\pi_{20}\sigma,$$

where π_{10}, α, ϕ, π_{20}, and σ are defined as in Section 4.2. This transducer exactly maps the sentences into their reduction structures. See Fig. 9.

Formal descriptions of these two-pushdown transducers may be deduced from those of the corresponding acceptors. They are omitted because they are lengthy and do not display any interesting property.

5.2. *The Second Modification*

Turning the nondeterministic two-pushdown transducers obtained above into parsers may be performed in essentially two different ways.

The *trial-and-error technique* consists in an exhaustive study of the different possible runs of the transducer. More precisely, each time n different instructions are applicable to a transducer, this transducer is replaced by n identical ones; the runs of these n transducers are then examined in parallel. The parser thus obtained determines the syntactical structures of any sentence presented to it; on the other hand, it does not necessarily halt when the string presented to it is not a sentence.[6]

The *selective technique* in principle works as follows: each time different instructions are applicable to the transducer a "selector" is appealed to; this selector, which has been built once for all for a given grammar, consults the input and/or pushdown tapes and deduces from it which instruction(s) is (are) to be applied, i.e., which instruction(s) lead to a syntactical structure. When different instructions are thus found, the different runs of the transducer are examined as in the trial-and-error technique; it should be noted that this case only occurs for ambiguous languages. In practical applications a selector consults only limited fractions of the tapes; the selective technique is then applicable to correspondingly limited classes of general phrase-structure grammars. The most difficult part in the construction of a selective parser is of course that of its selector; this problem will be briefly discussed in Section 6.2.

5.3. *Discussion*

Four parsing techniques have been described: the trial-and-error top-down, the trial-and-error bottom-up, the selective top-down and the selective bottom-up parsing technique. These techniques may be considered as basic but it is clear that their implementation allows for numerous variants some of which will be discussed in Section 5.5.

Clearly, the parsers described in Griffiths [1, 2] use the trial-and-error top-down technique. The parsers described in Eickel and Paul [3] and in Loeckx [18] are particular implementations of the selective bottom-up

[6] This corresponds to the fact that the language defined by a general phrase-structure grammar is recursively enumerable but not necessarily recursive.

technique. To our knowledge no parsers for general phrase-structure grammars have yet been described in the literature which use the trial-and-error bottom-up or the selective top-down technique.

5.4. *The Particular Case of Context-Free Grammars*

In the particular case of context-free grammars the two-pushdown acceptors and transducers may be systematically replaced by one-pushdown acceptors and transducers. Moreover the transducers may output the canonical generation (resp. reduction) sequences instead of the generation (resp. reduction) structures.

The distinction between the four parsing techniques roughly corresponds to the classification of parsers in e.g., Cheatham and Sattley [4], Knuth [5], Feldman and Gries [6]. As a difference the use of pushdown transducers provides a more precise description. The parsers described in, for instance, Brooker et al. [19], Cheatham and Sattley [4], Chartres and Florentin [20] use the trial-and-error top-down technique; those described in Irons [7] and Irons [21] use the trial-and-error bottom-up technique; those described in Greibach [22], Knuth [5] use the selective top-down technique; the parsers described in Knuth [23], Floyd [24], Irons [25], Wirth and Weber [26], Floyd [27], Gries [28] use the selective bottom-up technique.

5.5. *Implementation*

As already indicated the implementation of the parsing techniques described allows for many variants; in particular the selector of a selective parser may be implemented in different ways. In this Section some of these variants will be discussed; most of them may be considered as generalizations of existing implementations for context-free grammars.

A variant of the trial-and-error technique consists in examining the different runs of the transducers sequentially rather than in parallel. Clearly, this only works when a run of a transducer to which a sentence has been presented never leads into an infinite loop; in other words, a run of such a transducer should lead either to a final configuration (i.e., a configuration with the control unit in the final state and with the input and pushdown tapes empty) or to a dead-end (i.e., a configuration which is not a final configuration and to which no instruction is applicable). As this condition is not satisfied for all phrase-structure grammars, the variant is only applicable to a limited class of grammars. Note that this variant is used in most of the existing implementations of the trial-and-error technique for context-free grammars, an exception being Irons [21]; as there exist context-free grammars, viz., left recursive context-free grammars, for which a trial-and-error

top-down parser may enter an infinite loop, most of these parsers are provided with a device detecting these loops (e.g., Cheatham and Sattley [4], Chartres and Florentin [20].[7]

Another variant or, more precisely, a mixed version is obtained by providing a trial-and-error parser with some "selecting" features in order to reduce the number of possible runs to be examined. These selecting features could be generalizations of those used in parsers for context-free grammars (e.g., Irons [7]).

Selectors of selective parsers essentially differ from each other by the size of the fractions of the tape they consult (and consequently by the class of grammars to which they are applicable). Practically they could be obtained by generalizing those described in the literature for context-free grammars, for instance Knuth [23], Floyd [24] or Wirth and Weber [26]; the first of these parsers puts a fixed bound on the fraction of the input tape which may be consulted whereas the other two moreover put a fixed bound on the fraction of the pushdown tape. Note that the parsers for general phrase-structure grammars described in Eickel and Paul [3] and Loeckx [18] have been obtained in this way: They both constitute a generalization of the parser for context-free grammars described in Floyd [24].

6. Comments

6.1. *Comparison of Efficiency*

The efficiency of a parser is often defined as being inversely proportional to the number of "elementary actions" (e.g., transducer instructions) to be executed. Selective parsers are in this sense essentially more efficient than trial-and-error parsers.

On the other hand, top-down parsers and bottom-up parsers have the same efficiency; in fact, though the top-down and bottom-up acceptors associated with a given grammar have in general a different number of instructions, the number of instructions to be executed during the run leading to the final configuration is the same for both acceptors, as may be easily verified by comparing Figs. 7 and 9. In this respect it may be noted that Griffiths and Petrick are *not* comparing the efficiency of the top-down and the bottom-up parsing technique (Griffiths and Petrick [9], Griffiths and Petrick [10]); rather they are comparing the efficiency of some particular

[7] The possibility of constructing such a device is related to the fact that the language defined by a context-free grammar is recursive; it does not exist for general phrase-structure grammars.

selecting features added to a trial-and-error top-down and a trial-and-error bottom-up parser respectively.

6.2. *The Construction of Selective Parsers*

Without entering into details it is clear that the main difficulty in the construction of a selective parser consists in foreseeing all possible "situations" which may occur during parsing. More precisely it is necessary to study which are the possible contents of the input tape and the pushdown tapes. The property indicated in Section 4.1 (resp. 4.2) suggests that this study is related to the study of the strings of the canonical generation (resp. reduction) sequences.

On the other hand the two-pushdown generator, which is the dual of the top-down (resp. bottom-up) acceptor, generates the sentences of the language defined by the grammar with which the acceptor is associated. A slight modification makes it generate the strings of the canonical generation (resp. reduction) sequences of the sentences. The generator thus modified may then be used for the construction of a selector of a top-down (resp. bottom-up) parser. This idea is worked out for a particular parser in Loeckx [18]: the paper describes an algorithm which accepts an arbitrary phrase-structure grammar and constructs the selector of a bounded-context parser for it; the kernel of this algorithm is a slightly modified version of the bottom-up generator.

REFERENCES

1. T. V. GRIFFITHS, Turing machine recognizers for general rewriting systems, "Proc. of the Fifth Annual Symp. on Switching Circuit Theory and Logical Design," pp. 47–56, IEEE, New York, 1964.
2. T. V. GRIFFITHS, Some remarks on derivations in general rewriting systems, *Information and Control* **12** (1968), 27–54.
3. J. EICKEL AND M. PAUL, The parsing and ambiguity problem for Chomsky-languages, *in* "Formal Language Description Languages," (T. B. Steel, Ed.), pp. 52–75, North-Holland, Amsterdam, 1966.
4. T. E. CHEATHAM AND K. SATTLEY, Syntax-directed compiling, *in* "Proc. AFIPS 1964 Spring Joint Computer Conf.," Vol. 25, pp. 31–57, Spartan Books, Baltimore, 1964.
5. D. E. KNUTH, Top-down Syntax Analysis, Internal Report (Notes of a course given at the International Summer School on Computer Programming), California Institute of Technology, Pasadena, 1967.
6. J. FELDMAN AND D. GRIES, Translator writing systems, *Comm. ACM* **11** (1968), 77–113.
7. E. T. IRONS, A syntax-directed compiler for Algol 60, *Comm. ACM* **4** (1961), 51–55.

8. H. LANGMAACK AND J. EICKEL, Präzisierung der Begriffe Phrasenstruktur und strukturelle Mehrdeutigkeit in Chomsky-Sprachen, *in "3. Colloquium über Automatentheorie,"* (W. Händler et al., eds.), pp. 263–287, Birkhäuser Verlag, Basel, 1967.
9. T. V. GRIFFITHS AND S. R. PETRICK, On the relative efficiencies of context-free grammar recognizers, *Comm. ACM* **8** (1965), pp. 289–300.
10. T. V. GRIFFITHS AND S. R. PETRICK, Top-down versus bottom-up analysis, "Proceedings IFIP Congress 68," pp. B80–B85, North-Holland Publ. Co., Amsterdam, 1968.
11. N. CHOMSKY, On certain formal properties of grammars, *Information and Control* **2** (1959), 137–167.
12. A. V. AHO AND J. D. ULLMAN, The theory of languages, *Math. Systems Theory* **2** (1968), 97–125.
13. S. GINSBURG, "The Mathematical Theory of Context-Free Languages," McGraw-Hill Book Co., New York, 1966.
14. N. CHOMSKY, Context-free Grammars and Pushdown Storage, *in* "RLE Quart. Progr. Rep. No. 65," Research Lab. of Electronics, M.I.T., Cambridge, Mass., March 1962.
15. R. J. EVEY, Application of pushdown-store machines, Proceedings AFIPS 1963 Fall Joint Computer Conference, Vol. 24, pp. 215–227, Spartan Books, Baltimore, 1963.
16. P. C. FISCHER, Multi-tape and infinite-state automata, *Comm. ACM* **8** (1965), 799–805.
17. S. GINSBURG AND M. A. HARRISON, One-way nondeterministic real-time list-storage languages, *J. ACM* **15** (1968), 428–446.
18. J. LOECKX, "Mechanical Construction of Bounded-Context Parsers for Chomsky O-Type Languages," Doctoral thesis, University of Louvain, 1968; *Philips Res. Repts.* **24**, Suppl. No. 3, 1969.
19. R. A. BROOKER, I. R. MCCALLUM, D. MORRIS, AND J. S. ROHL, The compiler compiler, *in* "Annual Review in Automatic Programming," (R. GOODMAN, ed.), Vol. 3, pp. 229–271, Pergamon Press, New York, 1963.
20. B. A. CHARTRES AND J. J. FLORENTIN, An universal syntax-directed top-down analyzer, *J. ACM* **15** (1968), 447–464.
21. E. T. IRONS, An error-correcting parse algorithm, *Comm. ACM* **6** (1965), 669–673.
22. S. A. GREIBACH, Formal parsing systems, *Comm. ACM* **7** (1964), 499–504.
23. D. E. KNUTH, On the translation of languages from left to right, *Information and Control* **8** (1965), 607–639.
24. R. W. FLOYD, Bounded-context syntactic analysis, *Comm. ACM* **7** (1964), 62–67.
25. E. T. IRONS, Structural connections in formal languages, *Comm. ACM* **7** (1964), 67–72.
26. N. WIRTH AND H. WEBER, Euler: A generalization of Algol and its formal definition, *Comm. ACM* **9** (1966), 13–25, 89–99.
27. R. W. FLOYD, Syntactic analysis and operator precedence, *J. ACM* **10** (1963), 316–333.
28. D. GRIES, Use of transition matrices in compiling, *Comm. ACM* **11** (1968), 26–34.